STRCPY

The string copy library functions are vulnerable to buffer overflow attack.

Sean Barnum, Cigital, Inc. [vita[1]]

2007-04-17

Part "Original Cigital Coding Rule in XML"

Mime-type: text/xml, size: 11780 bytes

<table>
<tr><td><b>Attack Category</b></td><td>• Malicious Input
• Denial of Service</td></tr>
<tr><td><b>Vulnerability Category</b></td><td>• Buffer Overflow
• No Null Termination</td></tr>
<tr><td><b>Software Context</b></td><td>• String Management</td></tr>
<tr><td><b>Location</b></td><td></td></tr>
<tr><td><b>Description</b></td><td>The string copy library functions are vulnerable to buffer overflow attack.
strcpy() is the classic buffer overflow attack. Any variant of strcpy or any routine that behaves like it, copying a C-string from one buffer to another, is vulnerable to the same misuse and attack patterns.
The destination buffer must be big enough to hold the source string plus the null (\0) terminating character. Even if the destination buffer is large enough, there is a chance that the source buffer might not be null terminated and thus might overrun. Many of the string copy functions do not check buffer sizes and simply look for a null character to determine end of input. This gives an attacker opportunity to send input larger than the buffer size, overflowing the buffer. The attacker can exploit this to implement a denial of service (DoS) or buffer overflow attack.</td></tr>
<tr><td><b>APIs</b></td><td><table>
<tr><th>Function Name</th><th>Comments</th></tr>
<tr><td>_ftcscpy</td><td></td></tr>
<tr><td>_mbscpy</td><td>Windows</td></tr>
<tr><td>_tcscpy</td><td>Windows</td></tr>
<tr><td>lstrcpy</td><td></td></tr>
<tr><td>lstrcpyA</td><td></td></tr>
<tr><td>lstrcpyW</td><td>Windows</td></tr>
</table></td></tr>
</table>

[1] http://buildsecurityin.us-cert.gov/bsi-rules/35-BSI.html (Barnum, Sean)

mbscpy	AIX, Windows - copies multibyte character strings
olestrcpy	Windows
StrCpy	
StrCCpy	
StrCAdd	
StrCpyA	Windows
StrCpyW	Windows
ualstrcpy	This function provides unaligned UNICODE
ualstrcpyA	This function provides unaligned UNICODE
ualstrcpyW	This function provides unaligned UNICODE
wcscpy	Windows

Method of Attack

An attacker could force input of arbitrarily long strings to overrun the destination buffer of a strcpy() call or could potentially force input of an unterminated string as the source of a strcopy() call. Either way, a buffer overflow could occur.

A buffer overflow is most dangerous when arbitrary data can be used to overwrite the stack, heap, or other sensitive area of memory. When the boundaries of the destination buffer are overrun, the contents of the source buffer are copied into adjoining areas of memory. If the buffer is on the stack, the most common (and critical) attack point is the return address of the current subroutine. The attacker crafts an input string that contains a valid address. This address is specifically positioned so that when the destination buffer is overwritten, the address gets dropped on top of the original return address for the current routine. Execution continues, as normal, but when the routine attempts to return to the calling subroutine, it must read the return address from the stack. However, since the attacker overwrote that address with his or her own address, the routine jumps to the wrong place and suddenly the computer is under control of the attacker. If the attacker has crafted the attack properly, he or she will have other code waiting in the address he specified that can begin doing nasty things.

In a heap overflow situation, the attacker overwrites a buffer that is stored in heap memory. Similar address overwriting can occur, but this time the target address is commonly part of a "virtual jump

<table>
<tr><td></td><td>table" normally associated with C++ objects. Polymorphic objects commonly carry virtual function tables along with them that point to the routines associated with operating on the objects data. By overwriting that jump table, the attacker can control where the program will jump when one of those methods is called. This is becoming more common as the popularity of C++ increases for writing network-centric code.</td></tr>
<tr><td><b>Exception Criteria</b></td><td>There is usually no issue when used to copy const strings into variables.</td></tr>
<tr><td><b>Solutions</b></td><td>
<table>
<tr><th>Solution Applicability</th><th>Solution Description</th><th>Solution Efficacy</th></tr>
<tr><td>Any context where a string is to be copied.</td><td>Never use strcpy(). Replace strcpy() and similar routines with a bounded call. There are a variety of options for this.</td><td>Effective if correctly implemented. Choose the solution variant that makes it easiest to avoid careless errors.</td></tr>
<tr><td>On Windows</td><td>Replace the call strcpy(d, s) with the strsafe.h routine StringCbCopy(d, s, BUFFSIZE_D), which takes a buffer size in *bytes*, or StringCchCopy(d, s, BUFFSIZECHARS_D) which takes a buffer size in *characters*, which becomes important if you are using wide Unicode characters. When using Unicode, extra care must be taken to specify the buffer size using the correct units.</td><td>Effective if correctly implemented but requires caution when using Unicode.</td></tr>
</table>
</td></tr>
</table>

On BSD UNIX systems with strlcpy	Replace the call strcpy(d, s) with strlcpy(d, s, BUFFSIZE_D). This checks the bounds and prevents the buffer overrun. The strlcpy routine operates like strncpy() but takes care to always null terminate the destination string. This thwarts the attack where the string is exactly the size of the buffer.	Effective if correctly implemented.
Finally, on any remaining system (including any other UNIX)	At a minimum replace strcpy(d, s) with strncpy(d, s, BUFFSIZE_D). This will properly check the bounds and prevent strncpy() from overflowing the buffer. On systems with strlcpy(), use that as a replacement If the buffer d is allocated statically or on the stack, one can use sizeof(d) in place of BUFFSIZE_D. However, if d is a pointer to the heap, then sizeof(d) will not work and BUFFSIZE_D must be known	Effective if correctly implemented.

	through other means.	
Any context where a string is to be copied.	Consider banning all use of strcpy() by making it impossible to compile. In a common header for your code base, define the following: #define strcpy Unsafe_strcpy This way, if any developer attempts to use strcpy(), it will generate a compile error.	Effective at avoiding use of strcpy()
Any context where a string is to be copied.	As an absolute last resort, you can consider doing the following dynamic checks during runtime: To properly use lstrcpy() or any strcpy(), you must do the following: 1. Verify that dest is not NULL 2. Verify that strlen(source) < SIZE_OF_DEST 3. If using wide characters, SIZE_OF_DEST must be in correct units (i.e., # wide chars, not bytes) 4. Verify that source is null terminated	Effective but more complex and error-prone.

Signature Details	The strcpy() function is called.
Examples of Incorrect Code	`char str1[10];`

	`char str2[]="abcdefghijklmn";`
	`strcpy(str1,str2);`

Examples of Corrected Code	`/* If truncation is ok, the following works. */`
	`const int BUFFER_SIZE = 10;`
	`char str1[BUFFER_SIZE];`
	`char str2[]="abcdefghijklmn";`
	`/* in this case we know str1 isn't null, but in general we should check to confirm that. */`
	`/* strncpy() always works, but on systems such as Windows or BSD Unix, there are better choices. */`
	`strncpy(str1,str2, BUFFER_SIZE-1); /* limit number of characters to be copied */`
	`str1[BUFFER_SIZE-1] = '\0'; /`
	`* guarantee result will be null terminated */`
	
	`/* If truncation is unacceptable... */`
	`const int BUFFER_SIZE = 10;`
	`char str1[BUFFER_SIZE];`
	`char str2[]="abcdefghijklmn";`
	`/* verify buffer big enough to hold string and null termination */`
	`if ((str1 != 0) && (strlen(str2) < BUFFER_SIZE)) {`
	`strncpy(str1,str2, BUFFER_SIZE);`
	`} else {`
	`/* handle error */`
	`}`

Source References	• Viega, John & McGraw, Gary. *Building Secure Software: How to Avoid Security Problems the Right Way*. Boston, MA: Addison-Wesley Professional, 2001, ISBN: 020172152X • Howard, Michael & LeBlanc, David C. *Writing Secure Code, 2nd ed*. Redmond, WA: Microsoft Press, 2002, ISBN: 0735617228, pg. 82. • man page for strlcpy() • http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/resources/strings/usingstrsafe[2]
Recommended Resource	
Discriminant Set	**Operating System** • Any

Languages	- C - C++

Cigital, Inc. Copyright

The Build Security In (BSI) portal is sponsored by the U.S. Department of Homeland Security (DHS), National Cyber Security Division. The Software Engineering Institute (SEI) develops and operates BSI. DHS funding supports the publishing of all site content.

[1] mailto:copyright@cigital.com